PATENT
Docket No.: M4065.0340/P340
Micron Docket No.: 99-1243

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

APPLICATION FOR U.S. LETTERS PATENT

Title:

METHOD AND APPARATUS FOR CONNECTING A MASSIVELY PARALLEL PROCESSOR ARRAY TO A MEMORY ARRAY IN A BIT SERIAL MANNER

Inventor:

Graham Kirsch

Dickstein Shapiro Morin
& Oshinsky LLP
2101 L Street, N.W.
Washington, D.C. 20037
(202) 785-9700

METHOD AND APPARATUS FOR CONNECTING A MASSIVELY PARALLEL PROCESSOR ARRAY TO A MEMORY ARRAY IN A BIT SERIAL MANNER

BACKGROUND OF THE INVENTION

1. FIELD OF THE INVENTION

The present invention relates to the field of computer memory devices and, more particularly to the connection of a massively parallel processor array to a memory array in a bit serial manner to effect a byte wide data reorganization.

2. DESCRIPTION OF THE RELATED ART

The fundamental architecture used by all personal computers (PCs) and workstations is generally known as the von Neumann architecture, illustrated in block diagram form in Fig. 1. In the von Neumann architecture, a main central processing unit (CPU) 10 is used to sequence its own operations using a program stored in a memory 12. The memory 12, referred to herein as "main memory", also contains the data on which the CPU 10 operates. In modern computer systems, a hierarchy of cache memories is usually built into the system to reduce the amount of traffic between the CPU 10 and the main memory 12.

The von Neumann approach is adequate for low to medium performance applications, particularly when some system functions can be accelerated by special purpose hardware (e.g., 3D graphics accelerator, digital signal processor (DSP), video encoder or decoder, audio or music processor, etc.). However, the approach of adding accelerator hardware is limited by the bandwidth of the link from the

CPU/memory part of the system to the accelerator. The approach may be further limited if the bandwidth is shared by more than one accelerator. Thus, the processing demands of large data sets, such as those commonly associated with large images, are not served well by the von Neumann architecture. Similarly, as the processing becomes more complex and the data larger, the processing demands will not be met even with the conventional accelerator approach.

It should be noted, however, that the von Neumann architecture has some advantages. For example, the architecture contains a homogenous memory structure allowing large memories to be built from many smaller standard units. In addition, because the processing is centralized, it does not matter where the data (or program) resides in the memory. Finally, the linear execution model is easy to control and exploit. Today's operating systems control the allocation of system memory and other resources using these properties. The problem is how to improve processing performance in a conventional operating system environment where multiple applications share and partition the system resources, and in particular, the main memory.

One solution is to utilize active memory devices, as illustrated in Fig. 2, in the computer system. Put simply, active memory is memory that can do more than store data; it can process it too. To the CPU 10 the active memory looks normal except that it can be told to do something with the data contents and without the data being transferred to the CPU or another part of the system (via the system bus). This is achieved by distributing processing elements (PEs) in an array 14 through

out the memory structure, which can all operate on their own local pieces of memory in parallel. In addition, each PE 16 within the PE array 14 typically communicates with each other, as illustrated in Fig. 3, to exchange data. Thus, active memory encourages a somewhat different view of the computer architecture, i.e., "memory centered" or viewed from the data rather than the processor.

In a computer system having active memory, such as illustrated in Fig. 2, the work of the CPU 10 is reduced to the operating system tasks, such as scheduling processes and allocating system resources and time. Most of the data processing is performed within the memory 12. By having a very large number of connections between the main memory 12 and the processing resources, i.e., the PE array 14, the bandwidth for moving data in and out of memory is greatly increased. A large number of parallel processors can be connected to the memory 12 and can operate on their own area of memory independently. Together these two features can provide very high performance.

There are several different topologies for parallel processors. One example topology is commonly referred to as SIMD (single instruction, multiple data). The SIMD topology contains many processors, all executing the same stream of instructions simultaneously, but on their own (locally stored) data. The active memory approach is typified by SIMD massively parallel processor (MPP) architectures. In the SIMD MPP, a very large number of processors (usually a thousand or more) of relatively simple PEs are closely connected to a memory and organized so that each PE has access to its own piece of memory. All of the PEs

execute the same instruction together, but on different data. The instruction stream is generated by a controlling sequencer or processor.

The SIMD MPP has the advantage that the control overheads of the system are kept to a minimum, while maximizing the processing and memory access bandwidths. SIMD MPPs, therefore, have the potential to provide very high performance very efficiently. Moreover, the hardware consists of many fairly simple repeating elements. Since the PEs are quite small in comparison to a reduced instruction set computer (RISC), they are quick to implement into a system design and their benefit with respect to optimization is multiplied by the number of processing elements. In addition, because the PEs are simple, it is possible to clock them fast and without resorting to deep pipelines.

In one exemplary massively parallel processor array, each PE 16 in the PE array 14 uses only a single pin to connect to the memory 12. Thus, a one bit wide data connection is provided. When this is done, data is stored "bit serially" so that successive bits of a binary value are stored at successive locations in the memory 12. This storage format is referred to as "vertical" storage. Thus data read from and written to each PE will be read and stored, respectively, "vertically" in successive locations in the memory 12 as illustrated in Fig. 4. Thus, in Fig. 4, if each PE 16a – 16n in a row 22 of PE array 14 is an eight bit PE, i.e., it operates on eight bits of data at a time, the data in the memory will be stored in eight successive vertical locations as illustrated. As noted above, each PE is connected to memory 12 by a one bit wide data connection 24. Thus, data from PE 16c will be stored in a byte

sized area 20 of memory 12 in successive locations in area 20, i.e., it will be stored vertically as illustrated by arrow 30. The storage of data bit serially has a number of benefits. First, the number of data wires per PE 16 to the memory 12 is kept to a minimum. Second, it allows for variable precision arithmetic to be more easily and efficiently implemented. For example, ten, twelve, or fourteen bit numbers can be stored and processed efficiently. Third, in some cases, the difference in speed of the memory access versus the PE cycle time can be matched by serializing the data access.

There are some drawbacks, however, with storing the data from the PE array 14 bit serially. For example, in most applications, a chip containing a SIMD MPP array 14 and its associated memory 12 will have some form of off-chip interface which allows an external device, such as for example CPU 10 as illustrated in Fig. 2, to access the on-chip memory 12. CPU 10 sees data stored word-wide, i.e., "horizontally" as illustrated by arrow 32 in Fig. 4, referred to as normal mode. Thus, for external devices to access data stored vertically requires that the data be reorganized, i.e., converted, to the normal mode before being transferred from the memory to the external device, or converted by the external memory device before it can be used.

Converting between the two formats, i.e., normal and vertical, can be performed within the PE array 14 or within the external device that needs access to the data, but it would be more efficient to store the data in a single format, thus

avoiding having to store it in one format and convert it to another. Preferably, the single format would be the normal format used by the external devices.

Thus, there exists a need for a connection between a PE array and main memory in a MPP such that software data conversion is not required, and data can be stored in a normal mode or vertical mode in the memory.

SUMMARY OF THE INVENTION

The present invention provides a method and apparatus for connecting the processor array of an MPP array to a memory such that data conversion by software is not necessary, and the data can be directly stored in either a normal mode or vertical mode in the memory.

The above and other features and advantages of the invention are achieved by a connection circuit in which multiple PEs share their connections to multiple data bits in the memory array. Each PE is associated with a plurality of memory buffer registers, which stores data read from (or to be written to) one or two memory data bits. In horizontal (normal) mode connection the memory bits are selected so that all the bits of a given byte are stored in the same PE, i.e., each set of buffer registers associated with a respective PE contains one byte as seen by an external device. In vertical (bit serial) mode, each set of buffer registers contains the successive bits at successive locations in the memory corresponding to that PEs position in the memory word. The selection is achieved utilizing a multiplexer on the input to the register and a pair of tri-state drivers which drive each data line.

These and other advantages and features of the invention will become more readily apparent from the following detailed description of the invention which is provided in connection with the accompanying drawings.

BRIEF DESCRIPTION OF THE DRAWINGS

FIGURE 1 illustrates in block diagram form a conventional computer architecture;

FIGURE 2 illustrates in block diagram form the architecture of an active memory;

FIGURE 3 illustrates in block diagram form a conventional PE interconnect architecture;

FIGURE 4 illustrates vertical and horizontal data storage in a memory;

FIGURE 5 illustrates in schematic diagram form a connection between a PE array and a memory in accordance with the present invention;

FIGURE 6 illustrates in schematic diagram form an alternative embodiment for the connection between the PE array and memory of Fig. 5; and

FIGURE 7 illustrates in block diagram form a processor based system in which the present invention may be employed.

DETAILED DESCRIPTION OF PREFERRED EMBODIMENTS

The present invention will be described as set forth in the exemplary embodiments illustrated in Figs. 5-7. Other embodiments may be utilized and structural or logical changes may be made without departing from the spirit or scope of the present invention. Like items are referred to by like reference numerals.

In accordance with the present invention, a method and apparatus for connecting the processor array of an MPP to the memory such that data conversion by software is not necessary, and the data can be directly stored in either a normal mode or vertical mode in the memory is provided.

Fig. 5 illustrates the connection of a processor array of an MPP to a memory in accordance with the present invention. In accordance with the present invention, eight eight-bit PEs PE0-PE7 share their connections to 64 data bits in the memory array. A connection circuit 40a-40h is associated with each PE, i.e., PE0 to PE7, respectively. As illustrated in Fig. 5, each address Address0 – Address7 from the memory, such as for example memory 12 of Fig. 2, is an eight bit address, each having an associated eight bit data line bus 50a-50h. While the description of Fig. 5 is with respect to eight-bit PEs and eight bit data buses, it should be understood that the invention is not so limited and the invention is applicable to any data width, such as for example, ten bit, twelve bits fourteen bits, etc.

As shown in Fig. 5, each data bit line of data bus 50a-50h is connected to a second input of a respective multiplexer 52a-52h in each circuit 40a-40h associated

with each PE PE0 – PE7. Thus, the data bit line for the first bit of data, i.e., Bit0, is connected to the second input of multiplexers 52a-52h in the circuit 40a associated with PE0, the data bit line for the second bit of data, i.e., Bit1, is connected to the second input of multiplexers 52a-52h in the circuit 40b associated with PE1, and so on up to the data bit line for the last bit of data, i.e., Bit7, which is connected to the second input of multiplexers 52a-52h in the circuit 40h associated with PE7.

Referring back to the circuit 40a associated with PE0 in Fig. 5, the output from each multiplexer 52a-52h is connected to a respective buffer register 54a-54h. The output from each buffer register 54a-54h is connected to a respective input <0> - <7> of an eight-input multiplexer 60a. The output from each buffer register 54a-54h is also connected to the input of a respective pair of tri-state drivers 56a-56h. The output PE0 Din 62a from multiplexer 60 is connected to the first PE in the group of eight, i.e., PE0, by a single bit connection for passing data from the memory, i.e., Address0-Address7, to PE0. A second data line, PE Dout 64a, is also connected to the single bit connection to receive data from PE0 for writing to the memory into addresses Address0-Address7. The data line PE Dout 64a is connected to a first input of each multiplexer 52a-52h.

The output from the first and second tri-state drivers in pair 56a is connected to a respective bit data line of data bus 50a, i.e., the data bit line associated with the first bit of data Bit0 in Address0. In addition, the output from the second tri-state driver in pair 56a is connected to a third input of multiplexer 52a. The output from the first tri-state driver in pair 56b is connected to a respective data bit line of data

bus 50b, i.e., the data bit line associated with the first bit of data Bit0 in Address1, while the output from the second tri-state driver of pair 56b is connected to the second data bit line of data bus 50a and a third input of multiplexer 52b. The outputs from the remaining pairs of tri-state drivers 56c-56h are similarly connected, i.e., the output from the first tri-state driver of each pair 56c-56h is connected to its associated bit data line for the first bit of data Bit0 of data bus 50c-50h, while the output from the second tri-state driver of each pair 56c-56h is connected to a third input of its respective multiplexer 52c-52h and also connected to respective bit data lines of data bus 50a.

The above circuit 40a is substantially duplicated for each of the remaining circuits 40b-40h for each of the PEs in the group, i.e., PE1 – PE7, with the following exception. For the circuit 40b associated with PE1, the output from the first and second tri-state drivers in pair 56b is connected to a respective bit data line of data bus 50b, i.e., the bit data line associated with the second bit of data Bit1 from Address1, while the remaining pairs of tri-state drivers 56a and 56c-56h each have the output from the first tri-state driver connected to the bit data line associated with the second bit of data Bit1 of its associated data bus 50a and 50c-50h, respectively, and the output from the second tri-state driver is connected to respective bit data lines of data bus 50b and the third input of its respective multiplexer 52a and 52c-52h. For the circuit associated with PE2 (not shown), the output from the first and second tri-state drivers in pair 56c is connected to the bit data line associated with the third bit of data Bit2 of data bus 50c, while the remaining pairs of tri-state drivers 56a, 56b and 56d-56h each have the output from

the first tri-state driver connected to the data bit line associated with the third bit of data Bit2 of its associated data bus 50a, 50b and 50d-50h, respectively, and the output from the second tri-state driver is connected to a respective bit data line of data bus 50c and the third input to its respective multiplexer 52a, 52b and 52d-52h. This continues to the circuit 40h associated with the last PE in the group of eight PEs, i.e., PE7, where the output from the first and second tri-state drivers in pair 56h is connected to the data bit line associated with the last bit of data Bit7 in Address7 of data bus 50h, while the remaining pairs of tri-state drivers 56a-56g each have the output from the first tri-state driver connected to the data bit line associated with the last bit Bit7 of its associated data bus 50a-50g, respectively, and the output from the second tri-state driver is connected to a respective bit data line of data bus 50h and the third input to its respective multiplexer 52a-52g.

The operation of the circuitry as illustrated in Fig. 5 is as follows. Suppose for example a read of data is desired, and the data is stored in the memory in a vertical mode, i.e., data will be read from the memory 12 in a vertical fashion, as illustrated in Fig. 4, to each PE. Thus, it is desired to input each bit from a respective bit in Address0 to Address7 into a respective PE. For example, the first bit, i.e., Bit0, from each address Address0 to Address7 will be input to PE0, the second bit, i.e., Bit1, from each address Address0 to Address7 will be input to PE1, and so on up to the last bit, i.e., Bit7, from each address Address0 to Address7 which will be input to PE7. As the data is output on the data buses 50a-50h, each multiplexer 52a-52h will pass the data on its second input, i.e., data from the respective data buses 50a-50h, to its respective register 54a-54h. Thus, in circuit

40a, the first bit of data, i.e., Bit0, will be passed through multiplexers 52a-52h to registers 54a-54h, and then to multiplexer 60a. Multiplexer 60a will in turn send each bit of data Bit0 serially, i.e., from input <0> to input <7>, to PE0 via output 62a. Thus the output from multiplexer 60a to PE0 will be Bit0 from each of addresses Address0-Address7 i a serial manner.

Similarly, in circuit 40b, the second bit of data, i.e., Bit1, will be passed through multiplexers 52a-52h to registers 54a-54h, and then to multiplexer 60b. Multiplexer 60ba will in turn send each bit of data serially, i.e., from input <0> to input <7>, to PE1 via output 62b. The circuits associated with each remaining PE will operate similarly, up to circuit 40h, where the last bit of data, i.e., Bit7, will be passed through multiplexers 52a-52h to registers 54a-54h, and then to multiplexer 60h. Multiplexer 60h will in turn send each bit of data serially, i.e., from input <0> to input <7>, to PE7 via output 62h. Accordingly, the data is provided to each PE from the memory addresses Address0-Address7 in a vertical fashion.

Now suppose for example, a read is desired in which the data is stored in a horizontal mode in the memory, i.e., data stored in the memory in a normal mode (horizontal mode) as illustrated in Fig. 4, will be read from the memory and input to the respective PEs. Thus, each data bit from Address0 must be input to PE0 in a serial fashion from Bit0 to Bit7, each data bit from Address1 must be input to PE1 in a serial fashion from Bit0 to Bit7, and so forth. Referring to circuit 40a, as the data Bit0 to Bit7 from Address0 is provided on bus 50a, Bit0 on data bit line 0 of bus 50a will be input to the third input of multiplexer 52a, Bit1 on data bit line 1 of

bus 50a will be input to the third input of multiplexer 52b, Bit2 on data bit line 2 of bus 50a will be input to the third input of multiplexer 52c, and so forth up to Bit7 on data bit line 7 of bus 50a which will be input to the third input of multiplexer 52h. Multiplexers 52a-52h will pass the input on its third input to the respective registers 54a-54h. The data in registers 54a-54h will be sent to multiplexer 60a. Multiplexer 60a will in turn send each bit of data serially, i.e., from input <0> to input <7>, to PE0 via output 62a. Thus, PE0 will receive Bit0 to Bit7 from Address0 a single bit at a time.

Similarly, in circuit 40b, as the data Bit0 to Bit7 from Address1 is provided on bus 50b, Bit0 on data bit line 0 of bus 50b will be input to the third input of multiplexer 52a, Bit1 on data bit line 1 of bus 50b will be input to the third input of multiplexer 52b, Bit2 on data bit line 2 of bus 50b will be input to the third input of multiplexer 52c, and so forth up to Bit7 on data bit line 7 of bus 50b which will be input to the third input of multiplexer 52h. Multiplexers 52a-52h will pass the input on its third input to the respective registers 54a-54h. The data in registers 54a-54h will be sent to multiplexer 60b. Multiplexer 60b will in turn send each bit of data serially, i.e., from input <0> to input <7>, to PE1 via output 62b. Thus, PE1 will receive Bit0 to Bit7 from Address1 a single bit at a time.

The circuits associated with each remaining PE will operate similarly, up to circuit 40h, where as the data Bit0 to Bit7 from Address7 is provided on bus 50h, Bit0 on data bit line 0 of bus 50h will be input to the third input of multiplexer 52a, Bit1 on data bit line 1 of bus 50h will be input to the third input of multiplexer 52b,

Bit2 on data bit line 2 of bus 50h will be input to the third input of multiplexer 52c, and so forth up to Bit7 on data bit line 7 of bus 50h which will be input to the third input of multiplexer 52h. Multiplexers 52a-52h will pass the input on its third input to the respective registers 54a-54h. The data in registers 54a-54h will be sent to multiplexer 60h. Multiplexer 60h will in turn send each bit of data serially, i.e., from input <0> to input <7>, to PE7 via output 62h. Thus, PE7 will receive Bit0 to Bit7 from Address7 a single bit at a time. Accordingly, the data can be read from the memory in a horizontal mode.

Now suppose for example a write is desired in which the data from each PE will be stored in the memory in a vertical mode as illustrated in Fig. 4. Thus, it is desired to enter each of the eight bits from a PE into the same location in respective memory addresses Address0-Address7. Referring to circuit 40a of Fig. 5, the data will be serially output from PE0 on line PE Dout 64, which is connected to the first input of each multiplexer 52a-52h. The first bit of data output from PE0 will be passed by multiplexer 52a to register 54a, and then to the pair of tri-state drivers 56a. The first tri-state driver of pair 56a will pass the data to data bit line 0 of data bus 50a, which will write the data into the first bit Bit0 of Address 0. Similarly, the second bit of data output from PE0 will be passed by multiplexer 52b to register 54b, and then to the input of the pair of tri-state drivers 56b. The first tri-state driver of pair 56b will pass the data to data bit line 0 of data bus 50b, which will write the data into the first bit Bit0 of Address1. This continues for each bit of data from PE0, up to the last bit which is passed by multiplexer 52h to register 54h , and then to the input the pair of tri-state drivers 56h. The first tri-state driver of pair

56h will pass the data to data bit line 0 of data bus 50h, which will write the data into the first bit Bit0 of Address7.

The remaining circuits 40b-40h operate similarly to store the data into the respective location of each address Address0-Address7. For example, with respect to circuit 40b, the first bit of data output from PE1 will be passed by multiplexer 52a to register 54a, and then to the input of the pair or tri-state drivers 56a. The first tri-state driver of pair 56a will pass the data to data bit line 1 of data bus 50a, which will write the data into the second bit Bit1 of Address 0. Similarly, the second bit of data output from PE1 will be passed by multiplexer 52b to register 54b, and then to the input of the pair of tri-state drivers 56b. The first tri-state driver of pair 56b will pass the data to data bit line 1 of data bus 50b, which will write the data into the second bit Bit1 of Address1. This process continues for each bit of data from PE1 until the second data bit Bit1 of each address Address0-Address7 is filled.

Referring now to circuit 40h, the first bit of data output from PE7 will be passed by multiplexer 52a to register 54a, and then to the input of the pair of tri-state drivers 56a. The first tri-state driver of pair 56a will pass the data to data bit line 7 of data bus 50a, which will write the data into the last bit Bit7 of Address 0. Similarly, the second bit of data output from PE7 will be passed by multiplexer 52b to register 54b, and then to the input of the pair of tri-state drivers 56b. The first tri-state driver of pair 56b will pass the data to data bit line 7 of data bus 50b, which will write the data into the last bit Bit7 of Address1. This process continues for each

bit of data from PE7 until the last data bit Bit7 of each address Address0-Address7 is filled. Thus, data can be written from each PE to the memory in a vertical mode.

Now suppose for example a write is desired in which the data from each PE will be stored in the memory in a normal mode (horizontal mode) as illustrated in Fig. 4. Thus, it is desired to enter each of the eight bits from a PE sequentially into the respective bits of same address location. Referring to circuit 40a of Fig. 5, the data will be serially output from PE0 on line PE Dout 64, which is connected to the first input of each multiplexer 52a-52h. The first bit of data output from PE0 will be passed by multiplexer 52a to register 54a, and then to the input of the pair of tri-state drivers 56a. The second tri-state driver of pair 56a will pass the data to data bit line 0 of data bus 50a, which will write the data into the first bit Bit0 of Address 0. Similarly, the second bit of data output from PE0 will be passed by multiplexer 52b to register 54b, and then to the input of the pair of tri-state drivers 56b. The second tri-state driver of pair 56b will pass the data to data bit line 1 of data bus 50a, which will write the data into the second bit Bit1 of Address0. This continues for each bit of data from PE0, up to the last bit which is passed by multiplexer 52h to register 54h , and then to the input of the pair of tri-state drivers 56h. The second tri-state driver of pair 56h will pass the data to data bit line 7 of data bus 50a, which will write the data into the last bit Bit7 of Address0. Thus, the eight bits of data from PE0 will be written into Bit0 to Bit7 of Address0.

The remaining circuits 40b-40h operate similarly to store the data into the successive locations of each address Address1-Address7. For example, with respect

to circuit 40b, the first bit of data output from PE1 will be passed by multiplexer 52a to register 54a, and then to the input of the pair of tri-state drivers 56a. The second tri-state driver of pair 56a will pass the data to data bit line 0 of data bus 50b, which will write the data into the first bit Bit0 of Address1. Similarly, the second bit of data output from PE1 will be passed by multiplexer 52b to register 54b, and then to the input of the pair of tri-state drivers 56b. The second tri-state driver of pair 56b will pass the data to data bit line 1 of data bus 50b, which will write the data into the second bit Bit1 of Address1. This process continues for each bit of data from PE1 until the last bit of data from PE1 is written to Bit7 of Address1.

Referring now to circuit 40h, the first bit of data output from PE7 will be passed by multiplexer 52a to register 54a, and then to the input of the pair of tri-state drivers 56a. The second tri-state driver of pair 56a will pass the data to data bit line 0 of data bus 50h, which will write the data into the first bit Bit0 of Address7. Similarly, the second bit of data output from PE7 will be passed by multiplexer 52b to register 54b, and then to the input of the pair of tri-state drivers 56b. The second tri-state driver of pair 56b will pass the data to data bit line 1 of data bus 50h, which will write the data into the second bit Bit1 of Address7. This process continues for each bit of data from PE7 until the last data bit Bit7 is written to the last bit Bit7 of Address7. Thus, data can be written to the memory in a horizontal mode.

Thus, in accordance with the present invention, data can be read from a memory to a PE and written to the memory from the PE via a single bit connection in either a vertical or horizontal mode.

The use of a single register 52, such as for example 52a-52h, for each circuit 40a-40h allows only one byte to be held per PE in the memory buffer for either a read or a write operation. The use of a second register would allow write data to be held in one while the other is used for a read, or for data pipelining to be done for either. Fig. 6 illustrates in schematic diagram form an alternative embodiment of Fig. 5, in which each register 54a-54h of Fig. 5 is replaced by a pair of registers 80a and 80b. Accordingly, the output from a respective multiplexer 52, i.e., multiplexers 52a-52h of circuits 40a-40h of Fig. 5, is input to register 80a and register 80b. The output from each register 80a and 80b is input to a multiplexer 82. The output of multiplexer 82 is sent to a respective multiplexer 60, i.e., multiplexer 60a-60h of Fig. 5. Additionally, the output from registers 80a and 80b is input to a second muliplexer 84, whose output is connected to a respective pair of tri-state drivers 56, i.e., tri-state drivers 56a-56h of Fig. 5. The operation of the circuit as illustrated in Fig. 6 is similar to that as described with respect to Fig. 5, except that multiplexers 82 and 84 are used to determine from which register 80a or 80b data will be passed to either the input of tri-state driver pair 56 or multiplexer 60. Accordingly, the use of two registers 80a, 80b allows write data to be held in one while the other is used for a read, or for data pipelining to be done for either register.

An active memory device having the connection circuits 40a-40h of the present invention may be used in a processor-based system 300 of the type shown in Fig. 7. The processor-based system 300 comprises a processor 302 that

communicates with the memory device 312 and an I/O device 308 over a bus 320. It must be noted that the bus 320 may be a series of buses and bridges commonly used in a processor-based system, but for convenience purposes only, the bus 320 has been illustrated as a single bus. The memory device 312 includes connection circuits 40a-40h as previously described with respect to Figs. 5 and 6. The memory device 312 may be a SIMD MPP or any other type of DRAM or SRAM utilizing multiple PEs. In addition, the processor 302 may itself be an integrated processor which utilizes on-chip memory devices containing the circuitry of the present invention.

The processor-based system 300 may be a computer system, a process control system or any other system employing a processor and associated memory. The processor-based system 300 may also include read-only memory (ROM) 310 and may include peripheral devices such as a floppy disk drive 304 and a compact disk (CD) ROM drive 306 that also communicate with the processor 302 over the bus 320 as is well known in the art.

While the invention has been described in detail in connection with the preferred embodiments known at the time, it should be readily understood that the invention is not limited to such disclosed embodiments. Rather, the invention can be modified to incorporate any number of variations, alterations, substitutions or equivalent arrangements not heretofore described, but which are commensurate with the spirit and scope of the invention. Accordingly, the invention is not to be

seen as limited by the foregoing description, but is only limited by the scope of the appended claims.

5

10